DiMO: Differentiable Model Optimization and metaDiMO

Brando Miranda

August 2019

Abstract

We propose an AutoML paradigm that is end-to-end differentiable, versatile, scalable and by design optimizes to transfer to different data sets. The proposed method is scalable because it's fully differentiable and therefore trainable with Gradient Descent (GD). Consequently it does not rely on any Reinforcement Learning (RL) or Evolutionary methods. The proposed method is versatile because every part of the system can be easily learned or constrained with appropriate priors (e.g. biologically plausible priors). The pipeline is highly modular; the architecture search and its hyper parameters, optimizer and how credit is assigned all the way to performance predictor are all separate modules. This gives rise to rich combinations to experiment, deploy, train and constrain the system. Last but not least, the paradigm is formulated to consider multiple data sets at once and thus every module that has been learned is optimized to transfer to different tasks. Additionally, with a set of unknown data sets as part of the validation process, the system could potentially be trained to generalize to data sets it has not seen before.

1 Introduction

We propose an AutoML paradigm that is end-to-end differentiable, versatile, scalable and by design optimizes to transfer to different data sets. The proposed method is scalable because it's fully differentiable and therefore trainable with Gradient Descent (GD). Furthermore, its different from traditional approaches to AutoML because it does not rely on evolutionary or Reinforcement Learning (RL) techniques over discrete or non-differentiable search spaces which can take 2000 GPU days of RL training [1] or 3150 GPU days of evolution [2]. To make the system fully end-to-end two key components have to be made differentiable: first, the architecture search and second, the optimizer.

The method we propose uses convex relaxations over the discrete space of architecture. This method (although independently proposed from us) has already been proved effective in related work like: Differentiable Architecture Search (DARTS) [4]. In addition we also propose for the architecture search to include some hyper parameter search.

To make the optimizer differentiable one notices that iterative algorithms, like Stochastic Gradient Descent (SGD), are extremely simple Recurrent Neural Networks (RNNs). If we treat the gradient as a constant (e.g. data from the frame of reference to the RNN), then $W^{<t>} := W^{<t-1>} - \eta^{<t>} \nabla_W J(W^{<t-1>})$ is trivial RNN with only the subtraction operation and no memory cell. Making the optimization procedure trainable (independently proposed from us) with an RNN with memory has already been proven effective for inverse problem with Recurrent Inference Machines (RIM) [3] and therefore, seems likely to be easily applicable here.

The final piece for making the transger across different data sets is by taking an expectation over the data sets in question. This encourages the system to learn to propose architectures as a function of the data set. Furthermore, if one finds a semantic distributed representation of the data set (e.g. semantically aware embedding) and has a validation set of untested data sets (with a semantic representation of them) then one can have a system that proposes architectures to related data sets that it has not been trained on.

Now by connecting all the parts in a sequential manner we get a system that is fully differentiable and returns a fully trained neural networks with a custom optimizer for the set of data sets of interest.

2 Formulation

Consider the standard supervised learning paradigm with some fixed data set $d \in D$ summarized in one expression:

$$\min_{a \in A} ValErr[\operatorname*{argmin}_{m \in M(a)} Train(m)]$$

Note that to avoid cluttering of notation we suppressed the explicit dependence on the set of data points part of the training set and test set (but should be implicitly assumed). The above says that we searching for a model given by an architecture that minimizes the expected risk (approximated by test or validation error or doing a single epoch). This is usually done via two step optimization process: first choosing an architecture and then optimizing its weights with SGD on a train set. Usually the outer optimization (the architecture search) is done by humans and the inner search one is done automatically via SGD on the training set.

The above can also be framed using a function $g : D \rightarrow A$ that generates an architecture suggestion based on a data set $d \in D$ and thus restricts the model space search as follows:

$$\min_{g \in G} ValErr[\operatorname*{argmin}_{m \in M(g(d))} Train(m)] \tag{1}$$

where $g(d) = a$. In practice g is implemented by a human's biological neural networks that searches for the best architecture (usually for the fixed data set d).

The argmin in the previous formulation does not make it obvious how to implement this in an end-to-end fashion. Therefore, we will adopt the following notation change and view everything as a composition of functions instead of an minimization. To do that we first define the trainable $Train_{\theta_T}$ procedure as follows:

$$Train_{\theta_T}(g(d)) = \operatorname*{argmin}_{m \in M(g(d))} Train(m)$$

recall that to avoid cluttering of the notation the set of data points used for training are implicit in the notation but suggestive by the names of the function.

One can define the generator $g(d)$ as the composition of a feed-forward neural network $\hat{g}_{\theta_G}(d) = a_d$ and (for simplicity of exposition) a decoder $Dec_{\theta_D}(a_d)$ (though techniques shown in [4] can be used). Note that the decoder will also suggest the hyper parameters of the architecture. Therefore:

$$g(d) = Dec_{\theta_D}(\hat{g}_{\theta_G}(d))$$

Note that d can be represented as a one-hot vector (but eventually as a semantic trainable embedding) indicating which data set we are using. Note that the decoder outputs a soft one-hot vector representation of the architecture (e.g. using a softmax function).

Now consider defining $Train_{\theta_T}$ with an RNN implementing SGD (or a RIM [3]) and composing it with the output of the decoder:

$$Train(g(d)) = SGD_{\theta_S}(Dec_{\theta_D}(\hat{g}_{\theta_G}(d)))$$

Now with the above definition we can re-write expression 1 as follows:

$$ValErr[SGD_{\theta_S}(Dec_{\theta_D}(\hat{g}_{\theta_G}(d)))]$$

now because of function composition, we can simply optimize over each parameter separately using SGD:

$$\min_{\theta_S, \theta_D, \theta_G} ValErr[SGD_{\theta_S}(Dec_{\theta_D}(\hat{g}_{\theta_G}(d)))] \tag{2}$$

additionally to consider different data sets we take an expectation over the data sets in consideration as follows:

$$\min_{\theta_S, \theta_D, \theta_G} \mathbb{E}_{d \sim D}\left[ValErr[SGD_{\theta_S}(Dec_{\theta_D}(\hat{g}_{\theta_G}(d)))]\right] \tag{3}$$

we have arrived at a fully differentiable formulation for AutoML that considers transfer learning by design. Note that the distribution (and therefore importance) of data sets is up to the user. If learned it means the system learns which data set is best to focus on as to make the overall expected error small.

3 Training

We describe how training for expression 3 works. The system samples an architectures according to the architecture generator $g(d)$ and then computes the expectation in expression 3. After the forward pass if the system has been compute the weights of all the parameters are updated according to SGD:

$$\theta^{<t>} := \theta^{<t-1>} - \eta^{<t>} \nabla_\theta J(\theta^{<t-1>})$$

where $\theta = [\theta_S, \theta_D, \theta_G]$, $< t >$ indicates the iteration step and $J(\theta) = \mathbb{E}_{d \sim D}\left[ValErr[SGD_{\theta_S}(Dec_{\theta_D}(\hat{g}_{\theta_G}(d)))]\right]$. Note that SGD is induced by sampling batches of the validation and test set. Note one could sample the data set d. In addition it's easy to induces stochasticity in the optimizer by adding Gaussian noise, for example, when using Stochastic Gradient Descent Langevin (SGDL) [5] .

4 Remarks

- notice that the inner optimization no longer exists and the system is jointly optimized. Therefore, this formulation is not obviously equivalent to the two phase optimization that traditional supervised learning is based on.
- note that because SGD_{θ_S} can be implemented as a RIM [3], it can potentially be trained to minimizes the total loss over the training set at each step (see equation 8 from [3] for details).
- note that the $ValErr$ can be implemented by minimizing the whole validation set or random batches of it. It can be also implemented by a predictor performance function $Pred_{\theta_P}$. The predictor performance function can be pre-trained offline. Note that optimize θ_P requires care because doing it directly can promote the system to output a trivial solution when minimizing $J(\theta)$ like the zero function.
- notice that the parameters $\theta = [\theta_S, \theta_D, \theta_G]$ are optimized jointly using SGD.
- to promote the system to minimize the expected risk one can also restrict it to only go through one epoch when considering each model architecture $a = g(d)$.
- notice that the decoder can also be pre-trained using the auto-encoder paradigm. It takes embeddings of sampled architectures (or one-hot vectors) and tries to reconstruct them during the pre-training phase.
- a similar convex-relaxation of the architecture representation has already been proven successful [4] and thus is a part of the system that has already been tested, giving further confidence to this whole formulation.
- training the optimizer is also another part of the formulations that has already been shown successful for inverse problems [3] and thus has already been tested too.
- the expectation over the data set can be specified by the user thus the user can encode the importance of each dat apoint. Optionally it can be trivially learned with a softmax distribution and the system learns which data sets it should focus to minimize the expected error.
- as opposed to RIM [3] that uses gradients to learn how to do the updates, one can in fact learn directly the credit assignment function (instead of hard-coding back propagation). This can be very interesting the explore the wide space of alternative credit assignment algorithms and optimizers. Furthermore, one can encode different types of priors (e.g. SGD priors or biological priors) to bias the optimizer to explore specific update rules.

5 Conclusion

In conclusion we propose a promising framework to do AutoML where every part of the system can be learned. The system is more scalable than alternative methods because it uses SGD to be trained and avoids evolutionary and reinforcement learning (RL) approaches. In addition system is explicitly designed to learned a shared function g that learns to suggest architectures for a variety of selected data sets of interest. The method also chooses the best optimizer so to minimize the the validation loss. It's a promising method that automates the whole pipeline of supervised learning in a novel way.

6 Acknowledgements

I'd like to acknowledge Professor Sanmi Koyejo for his support and insightful discussions and questioning during the summer of 2019.

References

[1] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. CVPR, 2018.

[2] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. arXiv preprint arXiv:1802.01548, 2018.

[3] Patrick Putzky, Max Welling. Recurrent Inference Machines for Solving Inverse Problems. arXiv:1706.04008v1, 2017.

[4] Hanxiao Liu, Karen Simonyan, Yiming Yang. DARTS: Differentiable Architecture Search.

[5] Theory of Deep Learning III: Generalization Properties of SGD. Chiyuan Zhang, Qianli Liao, Alexander Rakhlin, Brando Miranda, Noah Golowich, Tomaso Poggio. CBMM Memo No. 067.